An Abstract Machine for Oz

Michael Mehl, Ralf Scheidhauer, and Christian Schulte

Programming Systems Lab
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{mehl,scheidhr,schulte}@dfki.uni-sb.de

Abstract. Oz is a concurrent constraint language providing for first-class procedures, concurrent objects, and encapsulated search. DFKI Oz is an interactive implementation of Oz competitive in performance with commercial Prolog and Lisp systems. This paper describes AMOZ, the abstract machine underlying DFKI Oz. AMOZ implements rational tree constraints, first-class procedures, local computation spaces for deep guards, and preemptive and fair threads.

1 Introduction

Oz is a concurrent constraint language [20, 19, 17, 6, 22] providing for functional, object-oriented, and constraint programming. It has a simple yet powerful computation model [19, 20], which extends the concurrent constraint model [10, 16] by first-class procedures, deep guards, concurrent state, and encapsulated search.

DFKI Oz [11] is an interactive implementation of Oz based on an incremental compiler and an abstract machine. It features a programming interface based on GNU Emacs, an object-oriented interface to Tcl/Tk, powerful interoperability features, a garbage collector, and support for stand-alone applications. Performance is competitive with commercial Prolog and Lisp systems.

This paper describes an abstract machine, called AMOZ, which covers important aspects of the DFKI Oz abstract machine. AMOZ implements rational tree constraints, first-class procedures, deep guards, and threads, leaving aside mutable state for objects [19], record constraints [21], as well as finite domain constraints and encapsulated search [18].

Constraint Store. By the very idea of concurrent constraint programming, computation emerges from adding constraints to a store. In this paper, we consider constraints over rational trees (as in Prolog II [5]) that enjoy a variable-centered normal form: adding constraints results in binding variables. This is utilized in AMOZ: binding variables triggers procedure application, reduction of conditionals, and readiness of threads.

First-class Procedures. Oz provides for first-class procedures typical of modern functional languages (e.g., Haskell [7], Scheme [3], and SML [12]). First-class procedures in Oz support higher-order functional programming [19], concurrent object-oriented programming [6], and encapsulated search [17]. In AMOZ, execution of a procedure definition dynamically creates a procedure (called closure in functional languages) and stores the procedure under a so-called name.
Procedure application is triggered by binding a variable to a name, from which the procedure to be applied is retrieved.

**Deep Guards.** Deep guards allow any expression in the guard of a conditional. Reduction of a deep guard is done in a local computation space. The main point of discussing deep guards here is to show implementation techniques for local computation spaces. Local computation spaces are needed to encapsulate search, and encapsulation of search is a must in a concurrent and reactive language. It is well known that the problem has not been solved in the Japanese Fifth Generation Project, leaving them with two incompatible language designs: concurrent logic programming and (constraint) logic programming. AKL was the first language that solved this problem, employing a design based on deep guards [8]. Oz, on the other side, employs a higher-order search combinator that uses local computation spaces but does not presuppose deep guards [17].

**Threads.** Languages like Prolog II and AKL have a single thread of control in which all computations are performed. However, this is insufficient for the fine grained concurrency found in concurrent constraint languages. Since general fairness does not seem practical, Oz provides for multiple threads that are scheduled fairly. In AMOZ threads are lightweight, implemented as multiple stacks of tasks that are scheduled preemptively and fairly.

The design of abstract machines for constraint based languages has been pioneered by the Warren Abstract Machine (WAM) [25, 1]. The implementation of DFKI Oz has been influenced by the AGENTS implementation of the concurrent constraint language AKL [8]. AKL is a deep guard language providing for encapsulated search. However, AKL does not provide for first-class procedures and threads. cc(FD) [23] is a constraint programming language specialized for finite domain constraints. It is a compromise between a flat and a deep guard language in that combinators (i.e., cardinality, disjunction, and implication) can be nested in guards, but procedure applications cannot. As AKL, it does not support first-class procedures and threads.

The paper is organized as follows. Section 2 gives an informal presentation of the computation model, and Sect. 3 gives an example. Section 4 shows unification for rational trees, and Sect. 5 introduces AMOZ. Threads and a limited case of conditional are introduced in Sect. 6. Section 7 extends the abstract machine for local computation spaces. Procedures are introduced in Sect. 8.

## 2 An Informal Computation Model

This section gives an informal presentation of the computation model underlying the sublanguage of Oz considered in this paper. A full description of Oz's computation model can be found in [20].

The notion of a *computation space* is central to the computation model. A computation space consists of a number of *tasks*¹ connected to a *store*.

¹ In other papers on Oz tasks are called actors.